嘟嘟独立博客

爱生活爱编码

search...

文章目录 隐藏目录

关闭阅读模式

Spring Boot干货系列：（八）数据存储篇-SQL关系型数据库之JdbcTemplate的使用

2017-04-13

Spring Boot干货系列 Spring Boot

前言

前面几章介绍了一些基础，但都是静态的，还不足以构建一个动态的应用。本篇开始就要介绍数据交互了，为了演示效果更加好，博主花了大把时间整合了一个后端模板框架，基于Bootstrap3的ACE模板，并实现了一个基本的增删改查分页功能。让我们一起动手，学技术的同时，顺便把我们的项目完善起来，这样跟着博主学到最后，你就有了一个属于自己的Spring Boot项目啦。

正文

本文介绍在Spring Boot基础下配置数据源和通过JdbcTemplate编写数据访问的示例。

添加依赖

这里需要添加spring-boot-starter-jdbc依赖跟mysql依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

数据源配置

在src/main/resources/application.properties中配置数据源信息。

```
spring.datasource.url = jdbc:mysql://localhost:3306/spring?useUnicode=true&characterEncoding=utf-8
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driver-class-name = com.mysql.jdbc.Driver
```

自定义数据源

spring-boot-starter-jdbc 默认使用tomcat-jdbc数据源，如果你想使用其他的数据源，比如这里使用了阿里巴巴的数据池管理,你应该额外添加以下依赖：

```
<dependency>
```

查看 17 条评论

```
    <artifactId>druid</artifactId>
    <version>1.0.19</version>
</dependency>
```

修改Application.java

```
@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }

        @Autowired
        private Environment env;

        //destroy-method="close"的作用是当数据库连接不使用的时候,就把该连接重新放到数据池中,方便下次使用调用.
        @Bean(destroyMethod =  "close")
        public DataSource dataSource() {
                DruidDataSource dataSource = new DruidDataSource();
                dataSource.setUrl(env.getProperty("spring.datasource.url"));
                dataSource.setUsername(env.getProperty("spring.datasource.username"));//用户名
                dataSource.setPassword(env.getProperty("spring.datasource.password"));//密码
                dataSource.setDriverClassName(env.getProperty("spring.datasource.driver-class-name'
                dataSource.setInitialSize(2);//初始化时建立物理连接的个数
                dataSource.setMaxActive(20);//最大连接池数量
                dataSource.setMinIdle(0);//最小连接池数量
                dataSource.setMaxWait(60000);//获取连接时最大等待时间，单位毫秒。
                dataSource.setValidationQuery("SELECT 1");//用来检测连接是否有效的sql
                dataSource.setTestOnBorrow(false);//申请连接时执行validationQuery检测连接是否有效
                dataSource.setTestWhileIdle(true);//建议配置为true，不影响性能，并且保证安全性。
                dataSource.setPoolPreparedStatements(false);//是否缓存preparedStatement，也就是PSCach
                return dataSource;
        }
}
```

ok这样就算自己配置了一个DataSource，Spring Boot会智能地选择我们自己配置的这个DataSource实例。

—— 脚本初始化 ——

```
CREATE DATABASE /*!32312 IF NOT EXISTS*/`spring` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `spring`;
DROP TABLE IF EXISTS `learn_resource`;

CREATE TABLE `learn_resource` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `author` varchar(20) DEFAULT NULL COMMENT '作者',
  `title` varchar(100) DEFAULT NULL COMMENT '描述',
  `url` varchar(100) DEFAULT NULL COMMENT '地址链接',
```

```
) ENGINE=MyISAM AUTO_INCREMENT=1029 DEFAULT CHARSET=utf8;

insert into `learn_resource`(`id`,`author`,`title`,`url`) values (999,'官方SpriongBoot例子','官方Spri
insert into `learn_resource`(`id`,`author`,`title`,`url`) values (1000,'龙果学院','Spring Boot 教程系
insert into `learn_resource`(`id`,`author`,`title`,`url`) values (1001,'嘟嘟MD独立博客','Spring Boot
insert into `learn_resource`(`id`,`author`,`title`,`url`) values (1002,'后端编程嘟','Spring Boot视频教
```

—— 开始使用JdbcTemplate ——

Spring的JdbcTemplate是自动配置的，你可以直接使用`@Autowired`来注入到你自己的bean中来使用。这里博主做了一套基本的增删改查操作。

实体对象 >

```
public class LearnResouce {
    private Long id;
    private String author;
    private String title;
    private String url;
        // SET和GET方法
}
```

Controller层 >

```
@Controller
@RequestMapping("/learn")
public class LearnController {
    @Autowired
    private LearnService learnService;
    private Logger logger = LoggerFactory.getLogger(this.getClass());

    @RequestMapping("")
    public String learn(){
        return "learn-resource";
    }

    @RequestMapping(value = "/queryLeanList",method = RequestMethod.POST,produces="application/js
    @ResponseBody
    public void queryLearnList(HttpServletRequest request ,HttpServletResponse response){
        String page = request.getParameter("page"); // 取得当前页数,注意这是jqgrid自身的参数
        String rows = request.getParameter("rows"); // 取得每页显示行数, ,注意这是jqgrid自身的参数
        String author = request.getParameter("author");
        String title = request.getParameter("title");
        Map<String,Object> params = new HashMap<String,Object>();
        params.put("page", page);
        params.put("rows", rows);
        params.put("author", author);
        params.put("title", title);
```

```
        List<Map<String, Object>> learnList=pageObj.getResultList();
        JSONObject jo=new JSONObject();
        jo.put("rows", learnList);
        jo.put("total", pageObj.getTotalPages());
        jo.put("records", pageObj.getTotalRows());
        ServletUtil.createSuccessResponse(200, jo, response);
    }
    /**
     * 新添教程
     * @param request
     * @param response
     */
    @RequestMapping(value = "/add",method = RequestMethod.POST)
    public void addLearn(HttpServletRequest request , HttpServletResponse response){
        JSONObject result=new JSONObject();
```

Service层 >

```
public interface LearnService {
    int add(LearnResouce learnResouce);
    int update(LearnResouce learnResouce);
    int deleteByIds(String ids);
    LearnResouce queryLearnResouceById(Long learnResouce);
    Page queryLearnResouceList(Map<String,Object> params);
}
```

实现类

```
@Service
public class LearnServiceImpl implements LearnService {

    @Autowired
    LearnDao learnDao;
    @Override
    public int add(LearnResouce learnResouce) {
        return this.learnDao.add(learnResouce);
    }

    @Override
    public int update(LearnResouce learnResouce) {
        return this.learnDao.update(learnResouce);
    }

    @Override
    public int deleteByIds(String ids) {
        return this.learnDao.deleteByIds(ids);
    }

    @Override
    public LearnResouce queryLearnResouceById(Long id) {
```

```
25
26      @Override
27      public Page queryLearnResouceList(Map<String,Object> params) {
28          return this.learnDao.queryLearnResouceList(params);
29      }
30  }
```

Dao层 >

```
1  public interface LearnDao {
2      int add(LearnResouce learnResouce);
3      int update(LearnResouce learnResouce);
4      int deleteByIds(String ids);
5      LearnResouce queryLearnResouceById(Long id);
6      Page queryLearnResouceList(Map<String,Object> params);
7  }
```

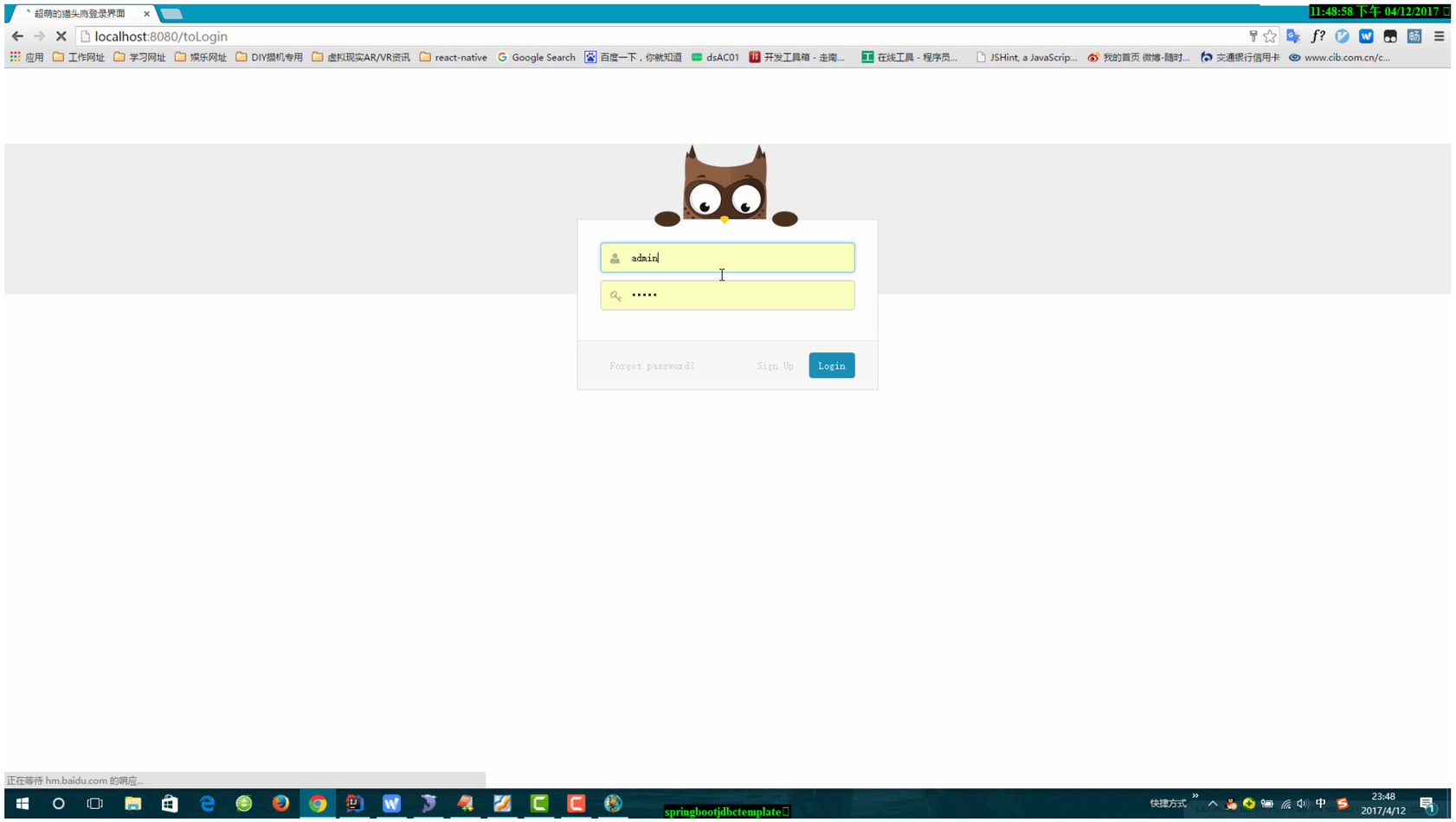
实现类,这里注入我们需要的JdbcTemplate

```
1  @Repository
2  public class LearnDaoImpl  implements LearnDao{
3      @Autowired
4      private JdbcTemplate jdbcTemplate;
5
6      @Override
7      public int add(LearnResouce learnResouce) {
8          return jdbcTemplate.update("insert into learn_resource(author, title,url) values(?, ?, ?)'
9      }
10
11     @Override
12     public int update(LearnResouce learnResouce) {
13         return jdbcTemplate.update("update learn_resource set author=?,title=?,url=? where id = ?'
14     }
15
16     @Override
17     public int deleteByIds(String ids){
18         return jdbcTemplate.update("delete from learn_resource where id in("+ids+")");
19     }
20
21     @Override
22     public LearnResouce queryLearnResouceById(Long id) {
23         List<LearnResouce> list = jdbcTemplate.query("select * from learn_resource where id = ?",
24         if(null != list && list.size()>0){
25             LearnResouce learnResouce = list.get(0);
26             return learnResouce;
27         }else{
28             return null;
29         }
30     }
```

```
33      public Page queryLearnResouceList(Map<String,Object> params) {
34          StringBuffer sql =new StringBuffer();
35          sql.append("select * from learn_resource where 1=1");
36          if(!StringUtil.isNull((String)params.get("author"))){
37              sql.append(" and author like '%").append((String)params.get("author")).append("%'");
38          }
39          if(!StringUtil.isNull((String)params.get("title"))){
40              sql.append(" and title like '%").append((String)params.get("title")).append("%'");
```

上面介绍的`JdbcTemplate`只是最基本的几个操作，更多其他数据访问操作的使用请参考：JdbcTemplate API

到此为止，后端交互代码都写好了，这里博主整合的bootstrap模板就不展示了，各位可以自行下载本篇对应的源码跑起来看看，效果很棒咯，如下：



总结

SpringBoot下访问数据库还是很简单的，只要添加依赖，然后在application.properties中配置连接信息。下一篇博主将介绍下Spring Boot对mybatis的整合。